Math 151b: Problem Set 3

Owen Jones

1/31/2024

Problem 1. (a) Using the Trapezoidal rule, we can approximate the area of the curve
$$\int_{t_n}^{t_{n+1}} y'(t)dt \approx (t_{n+1} - t_n)\frac{y'(t_{n+1}) + y'(t_n)}{2}.$$
We are given $f(t, y(t)) = y'(t)$ and $h = t_{n+1} - t_n$, so we substitute the expressions into our approximation to obtain
$\frac{h}{2}(f(t_{n+1}, y(t_{n+1})) + f(t_n, y(t_n)))$.
Assume $y_n = y(t_n)$ and let y_{n+1} be our approximation for $y(t_{n+1})$.
From (1), solving for $y(t_{n+1})$, we obtain
$y_{n+1} = y_n + \frac{h}{2}(f(t_{n+1}, y_{n+1}) + f(t_n, y_n))$.

(b) Using the one-step Forward Euler's method to approximate y_{n+1} in $f(t_{n+1}, y_{n+1})$ we obtain
$f(t_{n+1}, y_{n+1}) = f(t_{n+1}, y_n + hf(t_n, y_n))$.
Substituting into (2) we get
$y_{n+1} = y_n + \frac{h}{2}(f(t_{n+1}, y_n + hf(t_n, y_n)) + f(t_n, y_n))$.
Moreover, if we substitute $(3a)$ and $(3b)$ into our expression, we obtain
$y_{n+1} = y_n + \frac{h}{2}(k_2 + k_1)$.

(c) i. $f(t_{n+1}, y_n + hf(t_n, y_n)) = f(t_n + h, y_n + hf(t_n, y_n))$
Applying the chain rule:
$\frac{\partial f}{\partial h} = \frac{\partial f}{\partial t}\frac{\partial t}{\partial h} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial h} = f_t + ff_y$
$\frac{\partial^2 f}{\partial h^2} = \frac{\partial^2 f}{\partial t^2}(\frac{\partial t}{\partial h})^2 + \frac{\partial^2 f}{\partial y^2}(\frac{\partial y}{\partial h})^2 + 2\frac{\partial^2 f}{\partial t \partial y}\frac{\partial t}{\partial h}\frac{\partial y}{\partial h} = f_{tt} + f^2 f_{yy} + 2ff_{ty}$
Second order Taylor expanding around the point (t_n, y_n), we obtain
$f(t_n + h, y_n + hf(t_n, y_n))$
$= f + h(f_t + ff_y) + \frac{h^2}{2}(f_{tt}(\xi, \eta) + 2ff_{ty}(\xi, \eta) + f^2 f_{ty}(\xi, \eta))$

ii. Assume $y(t_n) = y_n$. $\tau_{n+1} = y(t_{n+1}) - y_{n+1}$. Taylor expand
$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(\xi')$
Using (i),
$y_{n+1} = y_n + \frac{h}{2}(2f + h(f_t + ff_y) + \frac{h^2}{2}(f_{tt}(\xi, \eta) + 2ff_{ty}(\xi, \eta) + f^2 f_{ty}(\xi, \eta)))$
$= y_n + hf + \frac{h^2}{2}(f_t + ff_y) + \frac{h^3}{4}(f_{tt}(\xi, \eta) + 2ff_{ty}(\xi, \eta) + f^2 f_{ty}(\xi, \eta))$
Thus, $y(t_{n+1}) - y_{n+1} = [y(t_n) - t_n] + h[y'(t_n) - f]$
$+ \frac{h^2}{2}[y''(t_n) - f_t + ff_y]$

$+ [\frac{h^3}{6}y'''(\xi') - \frac{h^3}{4}(f_{tt}(\xi,\eta) + 2ff_{ty}(\xi,\eta) + f^2 f_{ty}(\xi,\eta))]$

We know $y'(t) = f(t, y(t))$ and $y''(t) = f_t + ff_y$ and assume $y(t_n) = y_n$

Hence, $y(t_{n+1}) - y_{n+1} = [\frac{h^3}{6}y'''(\xi') - \frac{h^3}{4}(f_{tt}(\xi,\eta) + 2ff_{ty}(\xi,\eta) + f^2 f_{ty}(\xi,\eta))] = O(h^3)$, so τ_{n+1} is second order accurate.

Problem 2. If $y' = f(t, y) = 1 \Rightarrow k_i = 1, \quad \text{for } i = 1, 2, \cdots s$

Assume $y(t_n) = y_n$.

$\tau_{n+1} = y(t_{n+1}) - y_{n+1} = [y(t_n) - y_n] + h[y'(t_n) - \sum_{i=1}^{s} b_i k_i]$

$\Rightarrow \tau_{n+1} = y(t_{n+1}) - y_{n+1} = [y(t_n) - y_n] + h[1 - \sum_{i=1}^{s} b_i k_i]$.

If the numerical method is consistent, then it is at least first order accurate.

Thus, $h[1 - \sum_{i=1}^{s} b_i k_i] = 0 \Rightarrow 1 - \sum_{i=1}^{s} b_i k_i = 0$.

Because $k_i = 1, \quad \text{for } i = 1, 2, \cdots s, 1 - \sum_{i=1}^{s} b_i k_i = 1 - \sum_{i=1}^{s} b_i = 0$

$\Rightarrow \sum_{i=1}^{s} b_i = 1$.

If $\sum_{i=1}^{s} b_i = 1$, then $1 - \sum_{i=1}^{s} b_i k_i = 0$,

Thus, $h[y'(t_n) = f(t, y_n)] = 0$, so the numerical method is at least first order accurate.

Problem 3. (a) $\mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} x' \\ 2y' + x - \frac{\mu_0(x+\mu)}{r_1^3} - \frac{\mu(x-\mu_0)}{r_2^3} \\ y' \\ -2x + y - \frac{\mu_0 y}{r_1^3} - \frac{\mu y}{r_2^3} \end{bmatrix}$

$\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$

(c) I used about 20000 steps in my function.

(d) I chose an error tolerance of 10^{-7} as my absolute error tolerance. I used a max step size of 0.01, so RK45 used 1725 steps.

In [1]:
```
import numpy as np
import math as m
from matplotlib import pyplot as plt
from scipy import integrate
```

In [2]:
```
y_0=np.array([0.994,0,0,-2.00158510637908252240537862224])
points = 20000
t = np.linspace(0,17.1,points)
```

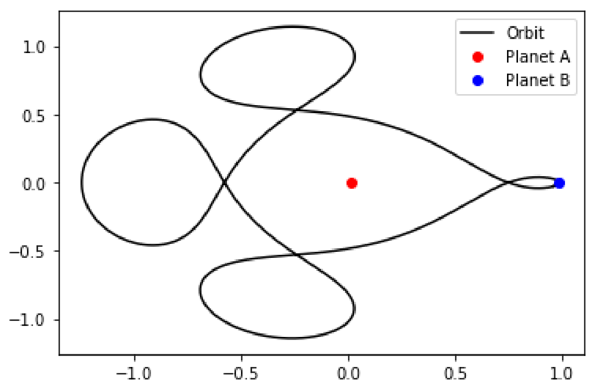
In [3]:
```
def f(t,y):
    mu=0.012277471
    mu_0=1-mu
    r_1=m.sqrt((y[0]+mu)**2+y[1]**2)
    r_2=m.sqrt((y[0]-mu_0)**2+y[1]**2)
    dx=2*y[3]+y[0]-mu_0*(y[0]+mu)/(r_1**3)-mu*(y[0]-mu_0)/(r_2**3)
    dy=-2*y[2]+y[1]-mu_0*y[1]/(r_1**3)-mu*y[1]/(r_2**3)
    return np.array([y[2],y[3],dx,dy])
```

In [4]:
```
def RK_method(f,y_0,t):
    y = np.zeros([len(t),4])
    y[0] = y_0
    for i in range(0,len(t)-1):
        h = t[i+1]-t[i]
        F1 = f(t[i],y[i])
        F2 = f((t[i]+h/2),(y[i]+F1*h/2))
        F3 = f((t[i]+h/2),(y[i]+F2*h/2))
        F4 = f((t[i]+h),(y[i]+F3*h))
        y[i+1] = y[i] + h/6*(F1 + 2*F2 + 2*F3 + F4)
    return y.transpose()
```

In [5]:
```
y_RK = RK_method(f,y_0,t)
```

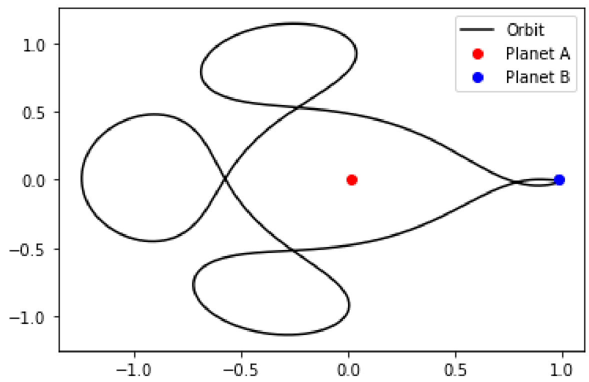
In [6]:
```
mu=0.012277471
plt.plot(y_RK[0],y_RK[1],'k-', label = "Orbit")
plt.plot(mu,0,'ro',label="Planet A")
plt.plot(1-mu,0,'bo',label="Planet B")
plt.legend()
```

Out[6]:
```
<matplotlib.legend.Legend at 0x7f90f8406940>
```



In [7]:
```
solution=integrate.solve_ivp(f,[0,17.1],y_0,max_step=0.01,atol=10e-7)
plt.plot(solution.y[0],solution.y[1],'k-',label = "Orbit")
plt.plot(mu,0,'ro',label="Planet A")
plt.plot(1-mu,0,'bo',label="Planet B")
plt.legend()
```

Out[7]:
```
<matplotlib.legend.Legend at 0x7f90f840e0d0>
```



Orbit
Planet A
Planet B
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0